



TITLE:

# サーチエンジンとソートエンジンの記号処理への応用 (情報の記憶と利用に関する理論的研究)

AUTHOR(S):

田中, 譲

---

CITATION:

田中, 譲. サーチエンジンとソートエンジンの記号処理への応用 (情報の記憶と利用に関する理論的研究). 数理解析研究所講究録 1981, 423: 323-343

ISSUE DATE:

1981-04

URL:

<http://hdl.handle.net/2433/102567>

RIGHT:

## サーチエンジンとソートエンジンの記号処理への応用

北海道大学 工学部

田 中 譲

### 1. 序論

データベース処理，特に2つ以上の関係に跨がる関係演算の高速化を目指し，著者等は2種類の機能モジュールを開発した<sup>(1)(2)</sup>．データベース処理においては，大量のデータの転送が隘路となる．この点を考慮して，2種類の機能モジュールを設計するにあたって，これらのモジュールにおけるデータ処理が，モジュールへのデータの転送，およびモジュールからのデータの転送と重畳され，同時実行されるように工夫した．2種類の機能モジュールは，サーチとソートというデータベース処理における2つの基本的な処理を行い，それぞれ，サーチエンジン，ソートエンジンと名付けられている。

サーチエンジンとソートエンジンは，各々のプロトタイプがTTLで組まれて完成している．プロトタイプのサーチエンジンは，16 bitの長さのキーを4095個，ソートされた順番でテーブルに格納しておき，これに流れ込んでくる任意個の探索キーの各々をテーブル内で探索し，見つけた場合

にはそのテーブルアドレスを出力し、見つからない場合には、そのキーより大きい最小のキーが入っているテーブルアドレスを出力する。几个の探索キーをテーブルアドレスに変換するのに  $n \mu\text{sec}$  (かかからない)。プロトタイプのソートエンジンは 16 bit のキー 4095 個が 1 個ずつ転送され、最後の 1 個の転送が終了するやいなや、これらをソートした順序で 1 個ずつ他のモジュールに転送し始めることができる。プロトタイプは、4095 個のソートを  $100 \text{ msec}$  で終了する。300  $\text{nsec}$  程度のメモリを用いて 2 つのモジュールを構成した場合、両者とも 1 つのデータあたり  $1 \mu\text{sec}$  以下の時間で各々の処理を行うことができる。

各々のモジュールは、ビットスライスのアーキテクチャになっているので、 $n$  個の同種のモジュールを横方向に結合することにより、 $n$  倍のワード長のデータのサーチとソートを行う回路を構成することができる。両モジュールとも、VLSI 化した際のピン数は数 10 ピン以内に制限することができる。上述の拡張性と共に、これらのモジュールが VLSI モジュールとして通していることを示している。

サーチエンジンとソートエンジンが VLSI 化されるとすると、これらの機能であるサーチとソートが基本演算と見做し得るようになり、種々のデータ処理に、新しいアルゴリズム

を提供するものと期待する。それは、このような機能モジュールにより、サーチとソートの時間複雑性が変化し、これにより、従来はサーチとソートを用いて処理することなど考えてもみられなかった問題を、2種類のモジュールを用いて処理することにより、今までのアルゴリズムより速く、スマートな処理方法を開発することが出来るかも知れないからである。

本論文では、2つのモジュールの記号処理への応用を目指し、この分野での基本的な問題の1つであるストリングマッチングを、上述のような問題の1つとして取り上げ、サーチエンジンとソートエンジンがこの種の問題にどのように応用され得るかを示す。本論文で示すストリングマッチングモジュールも核として、PROLOG等の論理型言語の実行を行うマシンを構成することも可能であると思われるが、これについては別の機会に報告する。このストリングマッチングのアルゴリズムでは、パターンをソートエンジンで前処理したものをサーチエンジンに格納し、複数のサーチエンジンで構成されるパイプに、サブジェクトストリングを先頭から順にデータの流れとして流し込むことにより処理する。これに関連して、本論文では可変長データのソートに関して新しいアルゴリズムを与える。

以下、第2章ではサーチエンジン（以下ではSEEと略記する）とソートエンジン（以下ではSOEと略記する）の機能の概説を行い、第3章では可変長データのソートアルゴリズムを、第4章ではストリングマッチングアルゴリズムを述べる。

## 2. サーチエンジン（SEE）とソートエンジン（SOE）の機能

SEEは値の順にソートされた $n$ 個の被探索キーからなるテーブル中で、別に与えられる $m$ 個の任意の順序に並んだ探索キーを順々に探索し、値の等しい被探索キーが見つかった時は、その旨を示すフラグと、値の等しい被探索キーの内、そのテーブル内アドレスの最小のもの、のアドレスを出力する。値の等しい被探索キーが見つからない時は、探索キーよりも大きい最小の被探索キーのテーブルアドレスの内、最小のものを出力する（Fig. 2.1）。実際のSEEによる処理は、探索キーを1個ずつ逐次にテーブルアドレスに変換するのでもなければ、すべての探索キーの入力が終了してからテーブルアドレスの出力を開始するのでもない。 $m$ 個の探索キーは長さ $m$ のデータの流れとしてSEEに流れ込み、SEEはこの流れを通す有限長のパイプとして働く。データの流れは、パイプから出るときにはフラグとテーブルアド

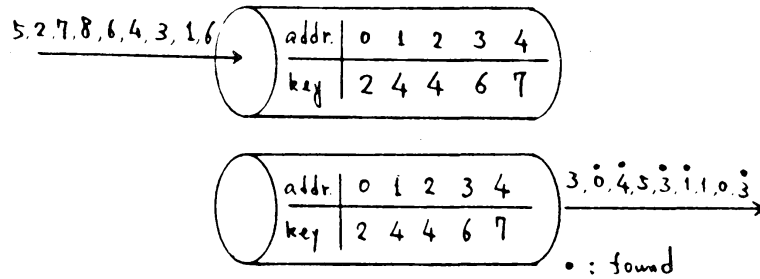
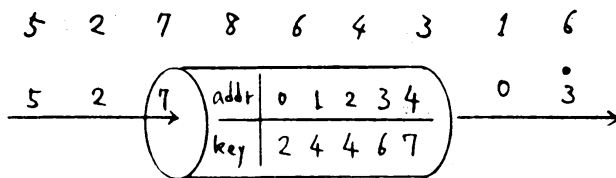


Fig. 2.1. SEEのサーチ機能

レスの流れへと変換されている (Fig. 2.2)。SEE のパイプの長さを SEE のレベル数といい、レベル数  $L$  の SEE は  $2^L - 1$  個の被探索キーからなるテーブルを格納することが出来る。

Fig. 2.2. SEEによるテーブルリスト  
リーム処理.

フラグが ON でテーブルアドレスが  $i$  のとき、これを  $i^*$  と表わすことにする。  $\mathbb{Z}$  を非負の整数の集合とし、  $\mathbb{Z}^*$  を

$$\mathbb{Z}^* = \{i, i^* \mid i \in \mathbb{Z}\},$$

$$\forall i \in \mathbb{Z} \quad i < i^* < i+1$$

と定義し、本論文ではこの数体系  $\mathbb{Z}^*$  を用いる。 $\mathbb{Z}^*$  から  $\mathbb{Z}$  への変換は存在し、例えば

$$\alpha: \mathbb{Z}^* \rightarrow \mathbb{Z}$$

$$\alpha(i) = 2i$$

$$\alpha(i^*) = 2i+1$$

とすればよい。以下ではフラグ出力とテーブルアドレス出力をまとめて  $\mathbb{Z}^*$  の要素の出力と見做したものを単にテーブルアドレス出力と言うことにする。SEE はテーブルアドレスの流れだけでなく、 $\sigma$  と  $\sigma$  の探索キーの流れも同期して出力するものと、SEE を Fig. 2.3 のように記号化する。テーブ

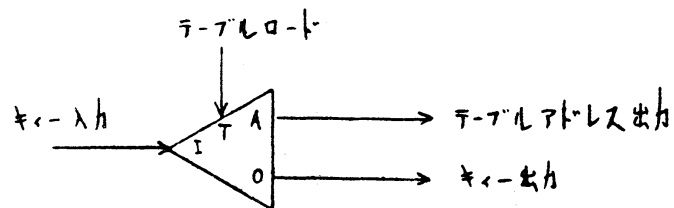


Fig. 2.3 SEEの記号

ル入力には、探索に先立って被探索キーを入力するための入口がある。この入力  $\sigma$ 、値の順にソートされたデータの流れが流入するのと同期して行われる。

SOEは勝手な順序で並んだ $n$ 個のキーからなるデータの流がSOEに流入され終わると同時に、この $n$ 個のキーを値の順序にソートしたデータの流を流出し始める (Fig. 2.4). SOEはすべてのキーのソートを行うので、キーのすべてがSOEに流入し終わる前にSOEからの流出が始まることはないので。

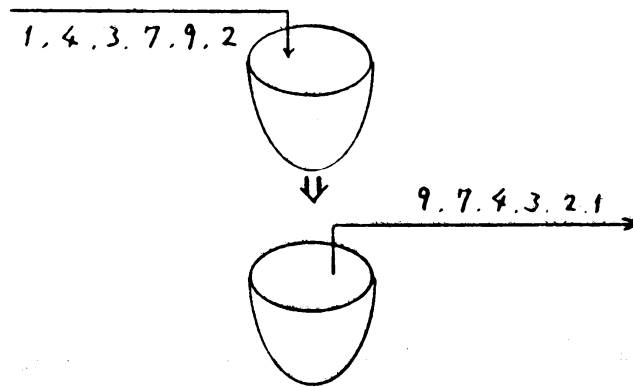


Fig. 2.4 SOE によるソート

SOEに $\ell$ レベル数が定義されており、レベル数が $L$ のSOEは、単独では最大 $2^L - 1$ 個のデータのソートを行う。SOEはFig. 2.5のように記号化する。

SEEとSOEとビットスライスアーキテクチャになっており、多倍長語のデータ処理するには各々を倍教だけ結合すればよい。<sup>(1)</sup>

サーチにおいて、ターゲットが小さくても1つのSEEには格納



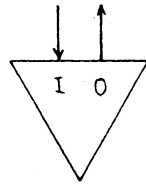


Fig. 2.5 SOE の記号化

いさなくなつたときは、被探索キーを順に  $2^L - 1$  個づつ、レベル数  $L$  の列々の SEE に格納し、探索キーはすべての SEE に流し込めばよい。データ数が多く、1 個の SOE ではリットいさな場合には、複数個の SOE をトータル接続と名付けた形に接続することにより、使用した SOE の個数倍のデータのリットが可能になる<sup>(1)</sup>。したがって、1 個の SEE、SOE のレベル数を  $L$ 、語長を  $w$  とすると、等面的には、 $L$ 、 $w$  を任意の正整数として、語長が  $Lw$  の、扱われるデータ数が単体の場合の  $2^L - 1$  に対して  $L(2^L - 1)$  個であるような SEE、SOE を構成することが出来る。

$2^L - 1$  語のメモリーの各番地  $i$  に、SEE に格納したデータの  $i$  番地のキーに対応するレコードを格納しておき、SEE の出力  $j^*$  に対してこのメモリーの  $j$  番地を読み出すことにすると、このシステムは全体として連想読み出しメモリーとなる (Fig. 2.6)。後置メモリーに、レコード部を上述のような順序でロードするためには、任意個の (キー、レコ

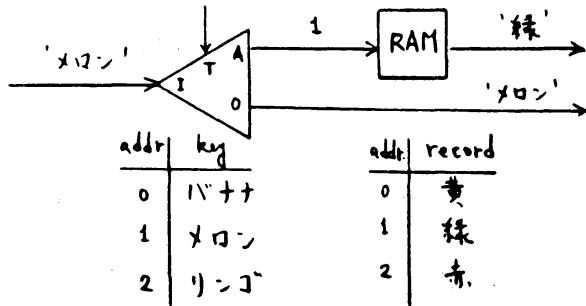
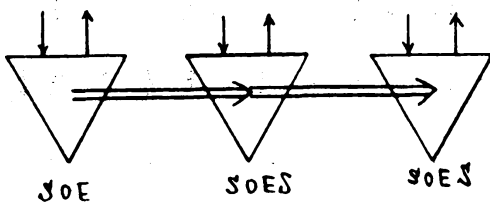
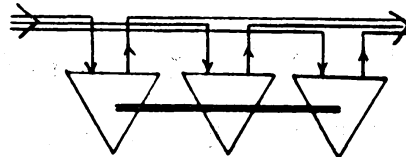


Fig. 2.6 SEEを用いた連想メモリー

ート)の対を、キーの値の順序に並べ替えなければならない。このような処理は、SOEと、このSOEにレベル数+1本の信号線が結合されるSOEシミュレータ(SOES)を用い、(キー、レコード)対の流れのキー部の流れとレコード部の流れを同期させて別々に各々SOEとSOESに流し込み、全部を流し込んだ後に、SOEからの出力とSOESからの出力を1個ずつ対にして出力すればよい。SOE, SOESの結合をFig. 2.7 (a)のように記号を表わす。先述したSOE, SEEの多倍長化のための結合はFig. 2.7 (b)のように表わす。



(a) SOESの記号表現



(b) SOEの多倍長化

Fig. 2.7. SEE, SOE, SOES の結合

## 3. 可変長データのリーク

$\Sigma$  を有限順序集合とする。1 を  $\Sigma$  の最小要素とし、 $\Sigma - \{1\}$  の要素の有限系列の集合を  $(\Sigma - \{1\})^*$  とする。 $(\Sigma - \{1\})^*$  の有限部分集合  $S$  の要素を辞書配列に従ってリークすることを考える。この処理は  $S$  の各要素に 1 を適当な個数だけ接続して各系列の長さを一定にして処理を行う (Fig. 3.1)。 $S$  を図

H	H 1 1 1
A J K L	A J K L
E F	E F 1 1
H I J K	H I J K
A B C	A B C 1
A B	A B 1 1
B	B 1 1 1

Fig. 3.1 可変長系列の集合  $S$  と  
 $S$  の要素の固定長化

固定長化したものを  $S_0$  で表わすことにする。 $S_0$  のリーク法として 3 種類の方法が考えられる。1 つの方法は語長を多倍長化した  $S_0E$  を用いることである。 $S_0E$  はビットスライスのアーキテクチャになっっているので充分な個数の  $S_0E$  を結合することにより、このような多倍長語の  $S_0E$  を構成できる。他の

2種類の方法では、無制限に多倍長化の機能を用いることなく、多倍長データのソートを行う。

アルゴリズム A は SOE と SOES を用いて、この処理をラティックスソートの考えに基づいて実行する (Fig. 3.2)。まず HLLL, AJKL, EFLl, HIJK, ABCl, ABLL, BLLl が最後尾の文字に関してソートされ、例えば EFLl, ABCl, ABLL, HLLL, BLLl, HIJK, AJKL となる。今度は、これを後から 2 番目の文字でソートする。この文字が等しい系列 a, b は、このときの系列の並びで a が b の前に並んでいれば a を b の前に出力するようにする。これを SOE を行うために、j の系列に順番を 0, 1, 2, ... と付け、後から 2 番目の文字とこの順番の連接をキーとしてソートする。この結果は EFLl, ABLL, HLLL, BLLl, ABCl, HIJK, AJKL の順になる。以下同様のことを後から 3 番目の文字、4 番目の文字、... と繰り返すことにより ABLL, ABCl, AJKL, BLLl, EFLl, HLLL, HIJK の順に出力され、ソートが完了する。

アルゴリズム B は SEE と SOE を Fig. 3.3 のように組み合わせ、この処理を実行する。このアルゴリズムでは、テーブルアドレス i, i\* を 1 文字と見做す。図中の  $\boxed{1/2}$  は 1 文字から 2 文字への変換器で、1 文字のデータが 2 個到着するのを

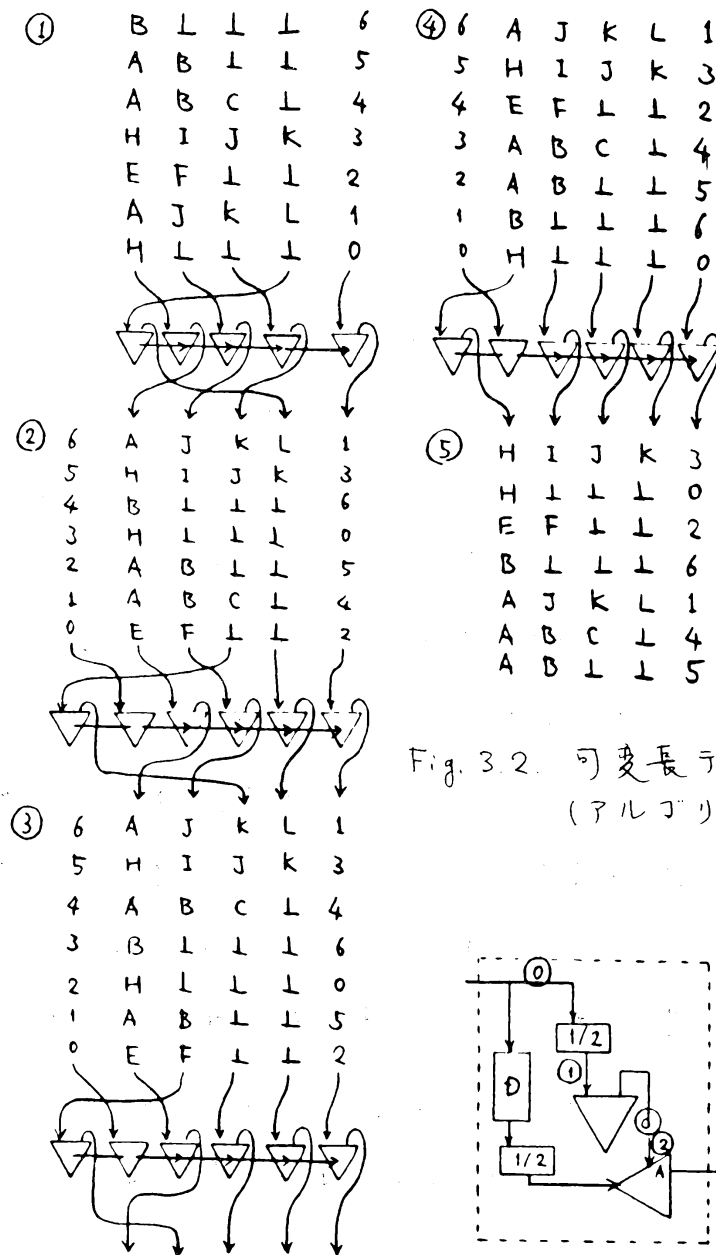


Fig. 3.2 可変長データのビット  
(アルゴリズム A)

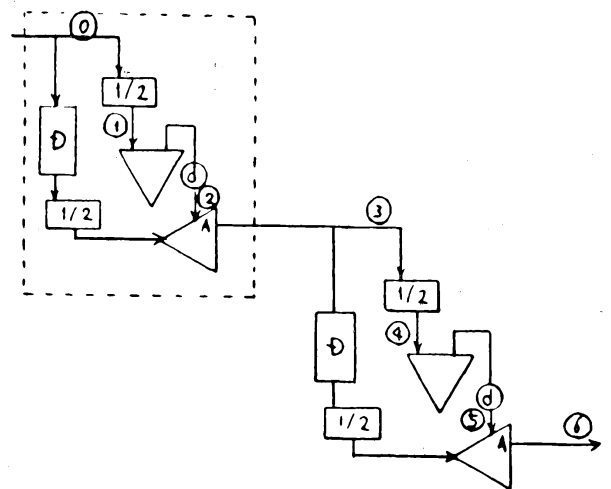


Fig. 3.3 可変長データのビット  
(アルゴリズム B)

待って、これらを2文字長のデータ1個に構成して出力する。  
 図中の④は、重複データの除去器である。図中に④の置  
 かれている場所では、データはソートされており、従って重  
 複データは連続する。よって重複データの除去は容易である。  
 S<sub>0</sub>のデータは各々2<sup>2</sup>文字であるとする。そうではないときは、  
 各データに1を適当な個数だけ連接すればよい。このとき、  
 S<sub>0</sub>のデータのソートを行うには、図中の破線や囲んだ部分  
 を1段用いればよい。物理的に破線や囲んだ部分を1個用意  
 する必要はなく、1個を繰り返し用いることが可能である。

例題のS<sub>0</sub>に対し、図中の⑤には HLLL AJKL EFLL HIJK  
 ABC L AB L L B L L L をデータの流れとして入力する。①では  
 これが2文字ずつまとめられ

(HL)(LL)(AJ)(KL)(EF)(LL)(HI)(JK)(AB)(CL)(AB)(LL)  
 (BL)(LL)

となる。これをソートし、重複の除去をしたものが②で、こ  
 れはテーブルとしてSEEに格納される。②とテーブルアドレ  
 スは

0 1 2 3 4 5 6 7 8 9  
 (LL)(AB)(AJ)(BL)(CL)(EF)(HL)(HI)(JK)(KL)

となる。図中の③は、SEEへのターゲットの格納が終了すると同時にSEEへの探索キークの入力が開始されるように時間遅れを調整するためのものである。SEEへの入力①と同じものである。従ってターゲットアドレス出力である③は、

$$6^* 0^* 2^* 9^* 5^* 0^* 7^* 8^* 1^* 4^* 1^* 0^* 3^* 0^*$$

となる。以下は同様で、④は、

$$(6^* 0^*)(2^* 9^*)(5^* 0^*)(7^* 8^*)(1^* 4^*)(1^* 0^*)(3^* 0^*)$$

となり、⑤とターゲットアドレスは

$$\begin{array}{cccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ (1^* 0^*) & (1^* 4^*) & (2^* 9^*) & (3^* 0^*) & (5^* 0^*) & (6^* 0^*) & (7^* 8^*) \end{array}$$

となり、⑥は、

$$5 \quad 2 \quad 4 \quad 6 \quad 1 \quad 0 \quad 3$$

となる。この並びは、J.をソートすると、HLLLは5+1番目に、次のAJKLは2+1番目に、...、ABLLは0+1番目に、そしてBLLLは3+1番目になることを示している。

#### 4. スtringマッチング

$\Sigma$  を有限順序集合とし,  $\Sigma^*$  を  $\Sigma$  の有限系列の集合とする。  
 $\Sigma^*$  の有限部分集合  $S$  をパターンの集合,  $t \in \Sigma^*$  をサブジェクト  
 スtring とする String マッチングとは, 各整数  $i > 0$   
 に対して  $t$  の  $i$  文字目から始まる部分 String で  $S$  に含  
 まれ, 長さの  $\rho_i$  と  $\rho$  長さ  $\rho$  のを見つける処理をいう (Fig. 4.1).

$$\Sigma = \{ \perp, A, B, C, \dots, Z \}$$

$$S = \{ C, CB, CBC, CJKL, FD \}$$

$$P_1 \quad P_2 \quad P_3 \quad P_4 \quad P_5$$

$$t = AFCBCJKLCA D$$

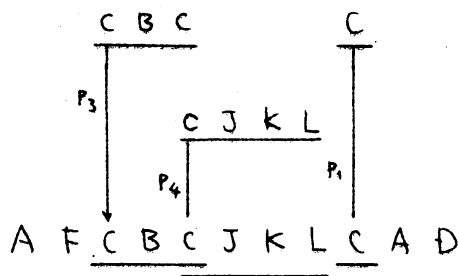


Fig. 4.1 String マッチングの例.

$m$  は  $S$  の最長要素長以上とし,  $S$  の各要素  $a$  に対して,  $\Sigma^*$   
 の要素の内  $\rho$  長さ  $\rho$  が  $m$  以下のものを辞書配列に並べたとき,  
 $a$  の前に並ぶ長さ  $m$  の系列  $\rho_i$  と  $\rho$  後に並ぶ  $\rho$  を  $a^-$  とし,



先頭部分が  $a$  であるような系列の内、最後に並んでゐるものを  $a^+$  と定義する。  $a^+$  の長さ  $m$  の系列となる。  $\Sigma$  を  $\Sigma$  の最小要素とすると、  $a = \perp \dots \perp$  のように  $\perp$  が任意個並んだ系列に対しては、  $a^-$  は存在しない。 Fig. 4.1 で  $m = 4$  とすると、  $C^- = BZZZ$ ,  $C^+ = CZZZ$ ,  $CB^- = CAZZ$ ,  $CB^+ = CBZZ$ ,  $CBC^- = CBBZ$ ,  $CBC^+ = CBCZ$ ,  $CJKL^- = CJKK$ ,  $CJKL^+ = CJKL$ ,  $FD^- = FCZZ$ ,  $FD^+ = FDZZ$  となる。  $S_1$  を

$$S_1 = \{ a^-, a^+ \mid a \in S \}$$

とし、  $S_1$  をソートしたものを  $S_2$  とする。 Fig. 4.1 の例では、順序集合は、

$$S_2 = \{ BZZZ, CAZZ, CBBZ, CBCZ, CBZZ, CJKK, CJKL, CZZZ, FCZZ, FDZZ \}$$

となる。

$\Sigma^*$  の要素の内、長さが  $m$  のものを  $\Sigma^m$  と表わす。  $\Sigma^m$  は辞書配列に従う順序集合である。各  $a \in S$  は、  $a^-$ ,  $a^+$  によつて  $\Sigma^m$  を最大3つの区間に分ける。

$u \in \Sigma^m$  に対して、

if  $u \leq a^-$  then  $u$  の先頭部分は  $a$  と一致しない。

if  $a^- < u \leq a^+$  then  $u$  の先頭部分が  $a$  に一致する.

if  $a^+ < u$  then  $u$  の先頭部分は  $a$  と一致しない.

のような関係がある。 $S_2$  の要素は順序集合  $\Sigma^m$  を Fig. 4.2 のように分割する。

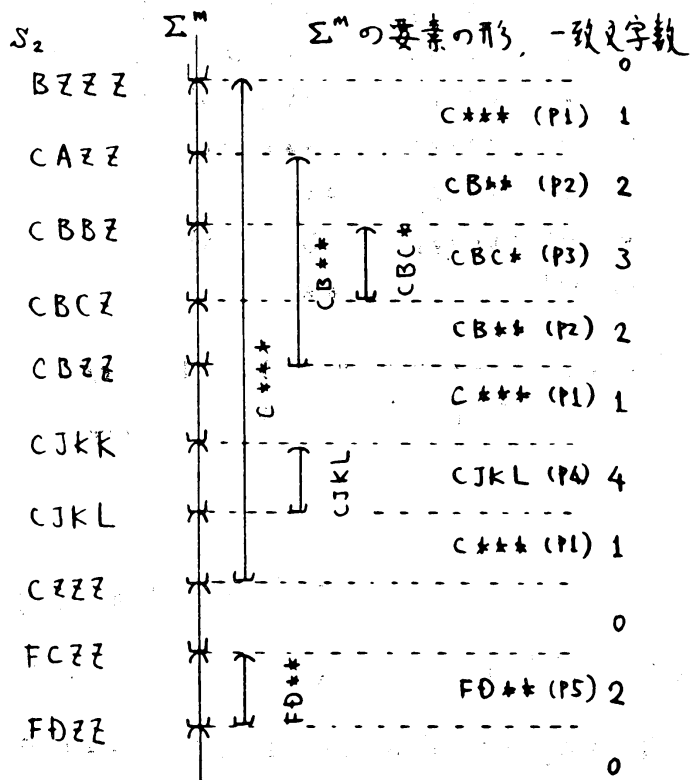


Fig. 4.2  $S_2$  による  $\Sigma^m$  の分割

後に, 4文字長の語表を持つ SEE があり, 例の  $S_2$  がこの SEE に格納されていて,  $u \in \Sigma^4$  が入力されたとする。このとき,  $u$  と出力されるテーブルアドレスには Fig. 4.3 のような関

係がある。

テーブルアドレス	u	一致文字数
0, 0* (BZZZ)	****	0
1, 1* (CAZZ)	C*** (P1)	1
2, 2* (CBBZ)	CB** (P2)	2
3, 3* (CBCZ)	CB* (P3)	3
4, 4* (CBZZ)	CB** (P2)	2
5, 5* (CJkk)	C*** (P1)	1
6, 6* (CJkL)	CJkL (P4)	4
7, 7* (ZZZZ)	C*** (P1)	1
8, 8* (FCZZ)	****	0
9, 9* (FDZZ)	FD** (P5)	2
10, 10*	****	0

Fig. 4.3  $S_2$  を u をサーチした際の  
出力アドレスと u の関係

m 文字長 (例では  $m=4$ ) の SEE を用いるかわりに、前章のアルゴリズム B と同様に、文字やテーブルアドレスを 2 つずつ組にして処理する。例えば BZZZ は BZ と ZZ に分割される。 $S_2$  の各要素をこのように分割したときできる 2 文字データの集合をリストし、重複を除去すると、例題の  $S_2$  に対しては、BZ, CA, CB, CJ, CZ, FC, FD, kk, kL, ZZ が得られる。この i 番目の 2 文字データを  $(i-1)^*$  の値にコード化する。その結果  $S_2$  の各要素は Fig. 4.4 (a) のようにコード化される。この

ードをさらに、 $2^k$ の要素を2個づつに区切り、同様のコード化を行う。このとき、 $m=2^k$ のストリングマッチングシステムは Fig. 4.4 のような構成で、SEE を1段連結したものである。処理の仕組みは図より明らかであろう。

このストリングマッチングシステムは、SEE のレベル数を  $L$  とするとき、表さが、

$$\log m \times L + (m-1)$$

のパイプと見做すことができ、サブジェクトストリングは文字の流れとして1文字ずつこのパイプに流れ込み、各文字が流出するときには Fig. 4.4 の出力部に示すようなストリングマッチングの情報が付加されて流れる。サブジェクトストリングが、このパイプ中を流れるだけで、一度に多くのパターンとの照合がとれる点が、本システムの特徴である。(Fig. 4.5)

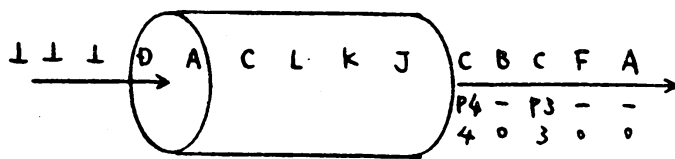


Fig. 4.5 ストリングマッチング  
モジュールのスリーム処理

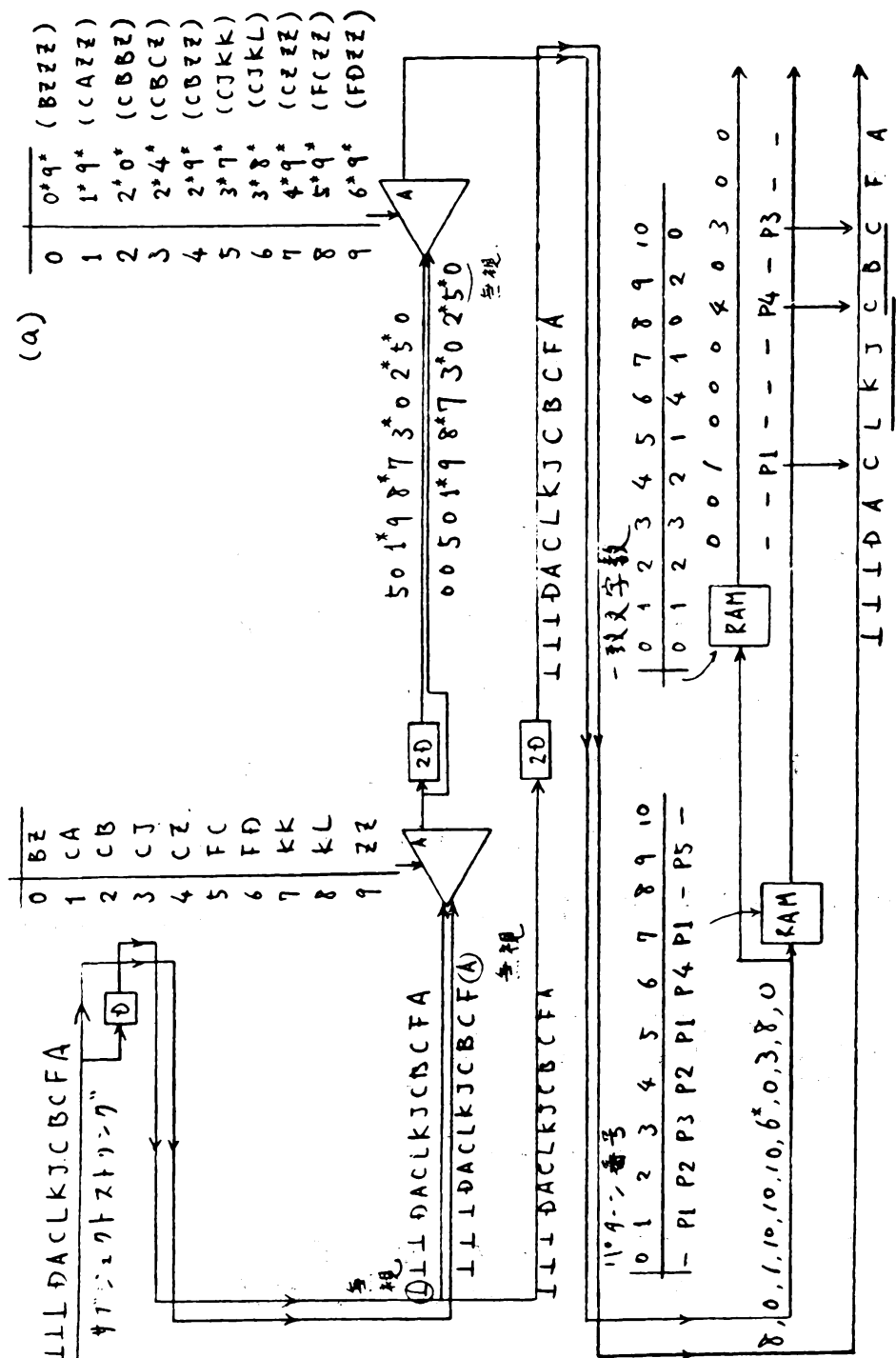


Fig. 4.4. SEEを用いた構成（たストリングマッチングモジュール）

## 5. 結論.

データベースの高速処理を目指して開発したSER, SOEが拡張性と実現性の両方において, VLSI化に向けたアーキテクチャであることを述べ, これらがVLSI化されることにより, サーチとソートを基本演算と見做し得るようになった場合に, どのような効果が期待され得るかを, ストリングマッチングを例にとり示した。さらに高度な応用として, 論理型言語の推論機構部への応用等を検討中である。

本論文で提案したストリングマッチングシステムは, Fig. 4.5のような1つの機能モジュールと見做すことができ, これを組み合わせるにより, ワードプロセッシングやトキメントプロセッシングへの応用が期待される。

## 参考文献

- (1) Y. Tanaka et al., "Pipeline Searching and Sorting Modules as Components of a Data Flow Database Computer," Proc. IFIP Congress 80, 1980 (Tokyo), pp. 429-432.
- (2) 田中 譲, 'データストリーム処理方式のデータベースシンセシス', "情報処理研究, 記号処理 12-14, 1980, pp. 97-103.